PUC Minas

Linguagens de Programação (Ciência da Computação)

Professora: M. Sc. Luciana De Nardin
luciana@pucpcaldas.br

1 Especificação de LP

Sintaxe

Forma das expressões, das instruções e das unidades do programa

Semântica

Significado das expressões sintaticamente corretas

- **Sintaxe**
  - Linguagem bem projetada
    - Semântica deve seguir-se diretamente da sintaxe
    - A forma da instrução deve sugerir o que ela realiza
    - Descrever a semântica >> BNF 
    - Descrever a semântica >> sem padrão 

- **Sintaxe**
  - Linguagem (natural - inglês ou artificial - java)
    - Conjunto de sequências de caracteres de um alfabeto >> sentenças
  - Regras da sintaxe
    - Quais sequências de caracteres do alfabeto estão nela

- **Sintaxe**
  - Exemplo:
    - Todo nome só pode conter letras e dígitos
    - “_” é considerado uma letra
    - Primeiro caracter deve ser letra
    - Palavras reservadas não podem ser usadas como nomes de variáveis

- **Sintaxe**
  - Unidades léxicas
    - Lexemas: identificadores, literais, palavras especiais…
    - Programas: sequência de lexemas (e não caracteres)
    - Token: categoria dos lexemas

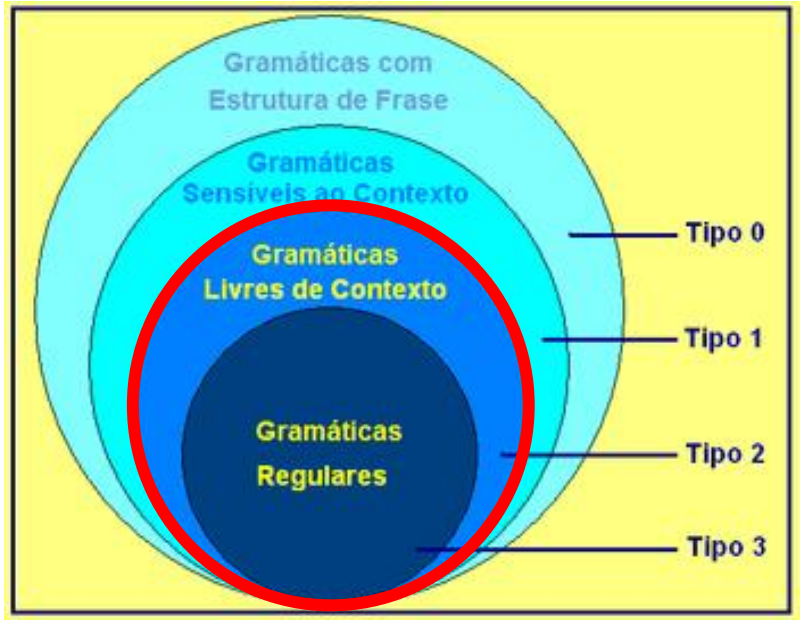
- **Sintaxe**
  - Unidades léxicas
    - Tokens *vs.* lexemas

A = 2 * B + 17;

Lexemas	Símbolos (token)
A	identificador
=	sinal_igual
2	int_literal
*	mult_op
B	identificador
+	soma_op
17	int_literal
;	ponto_e_virgula

Diferença?

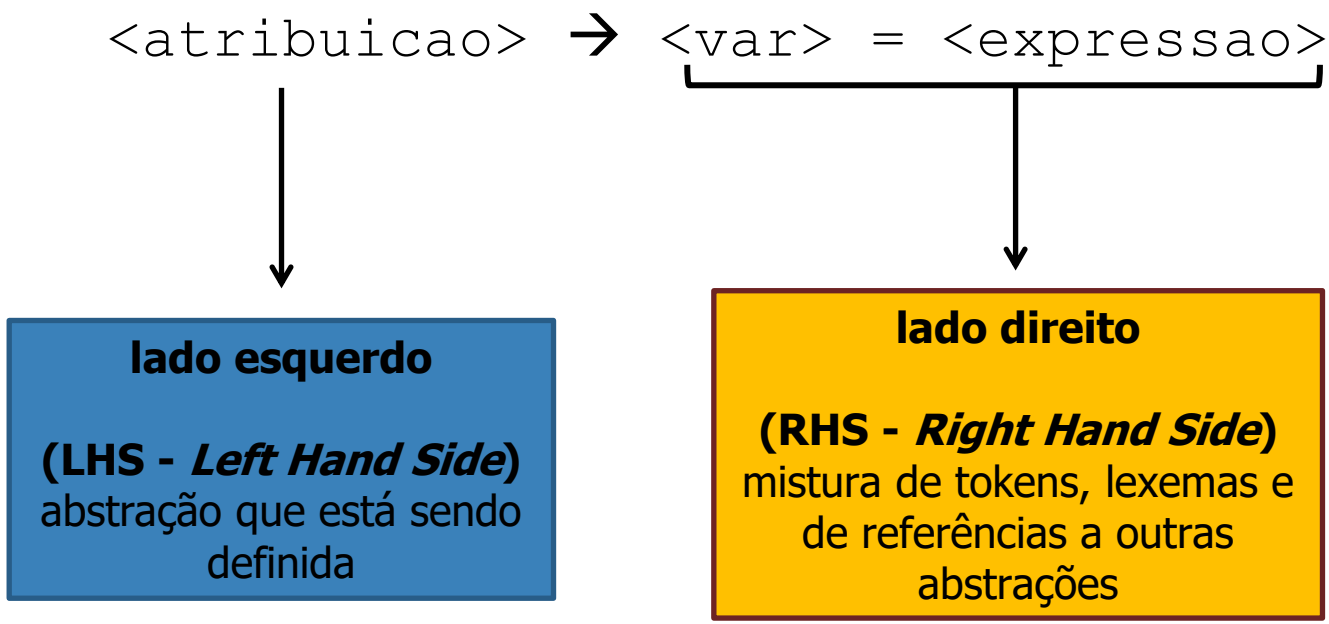
- **Sintaxe**
  - Forma de Backus-Naur



- **Sintaxe**
  - Origem BNF
    - 1959: John Backus (documentação Algol 60)
    - 1960: Peter Naur (Algol 60)
    - 1960: forma de Backus-Naur ou BNF
    - BNF é quase idêntica a GLC

- **Sintaxe**
  - Fundamentos da BNF
    - Metalinguagem = linguagem para descrever outras linguagens
    - Usa abstrações para definer estruturas estáticas

Sintaxe


<atribuicao> → <var> = <expressao>
lado esquerdo
(LHS - Left Hand Side)
abstração que está sendo definida
lado direito
(RHS - Right Hand Side)
mistura de tokens, lexemas e de referências a outras abstrações

Sintaxe

<atribuicao> → <var> = <expressao>

<var> = <expressao> é a definição de <atribuicao>

Atribuições >> símbolos não terminais

Tokens ou lexemas >> símbolos terminais

<total> = <operando1> + <operando2>

- **Sintaxe**
  - Gramáticas e derivações
    - Símbolo de início: não terminal e representa um programa
    - Derivação: geração de cada sentença
    - Forma sentencial: cada string de derivação

- **Sintaxe**
  - Exemplo

```
<programa> → begin <lista_inst> end
<lista_inst> → <inst>
               | <inst>;<lista_inst>
<inst> → <var> = <expressao>
<var> → A | B | C
<expressao> → <var> + <var>
              | <var> - <var>
              | <var>
```

→ ou ::= ou = > lê-se como “deriva”

```
<programa> → begin <lista_inst> end
<lista_inst> → <inst>
             | <inst>; <lista_inst>
<inst> → <var> = <expressao>
<var> → A | B | C
<expressao> → <var> + <var>
            |  <var> - <var>
            |  <var>
```

Instrução `begin A = B + C ; B = C end` é válida???

```
<programa> => begin <lista_inst> end
           => begin <inst>; <lista_inst> end
           => begin <var> = <expressao> ; <lista_inst> end
           => begin A = <expressao> ; <lista_inst> end
           => begin A = <var> + <var> ; <lista_inst> end
           => begin A = B + <var>; <lista_inst> end
           => begin A = B + C ; <lista_inst> end
           => begin A = B + C ; <inst> end
           => begin A = B + C ; <var> = <expressao> end
           => begin A = B + C ; B = <expressao> end
           => begin A = B + C ; B = <var> end
           => begin A = B + C ; B = C end
```

S ::= <cálculo>

<cálculo> ::= <expressão> =

<expressão> ::= <valor>

| <valor> <operador> <expressão>

<valor> ::= <número> | <sinal> <número>

<número> ::= <semsinal> | <semsinal>. <semsinal>

<semsinal> ::= <dígito> | <dígito> <semsinal>

<dígito> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<sinal> ::= + | -

<operador> ::= + | - | * | /

Criação de uma Expressão Válida

S ::= <cálculo>
<expressão> =
<valor> <operador> <expressão> =
<número> <operador> <expressão> =
<semsinal> <operador> <expressão> =
<dígito> <semsinal> <operador> <expressão> =
2 <semsinal> <operador> <expressão>
2 <digito> <operador> <expressão> =
25 <operador> <expressão> =
25 * <expressão> =
25 * <valor> =
25 * <número> =
25 * <semsinal> . <semsinal> =
25 * <dígito> . <semsinal> =
25 * 1. <semsinal> =
25 * 1. <dígito> =
25 * 1.5 =

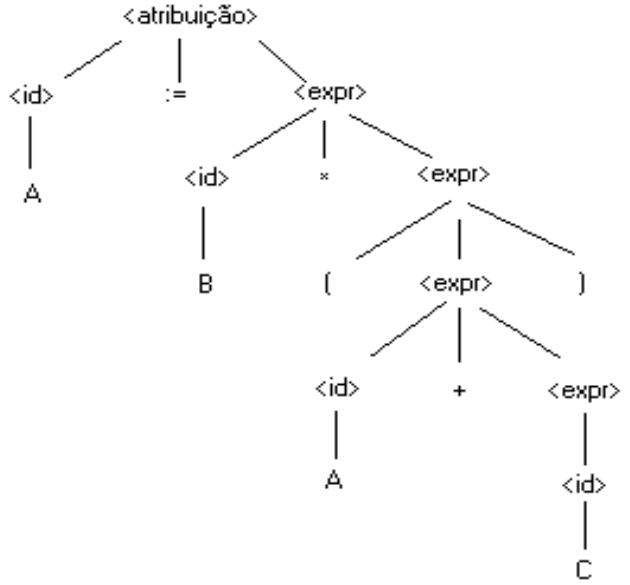
- **Sintaxe**
  - Exercício 1

```
<atribuição> → <id> = <expr>
<id> → A | B | C
<expr> → <id> + <expr>
        | <id> * <expr>
        | (<expr>)
        |  <id>
```

As instruções `C = (B + (C * A))` e `A = B * (A + C)` são válidas para a gramática acima?

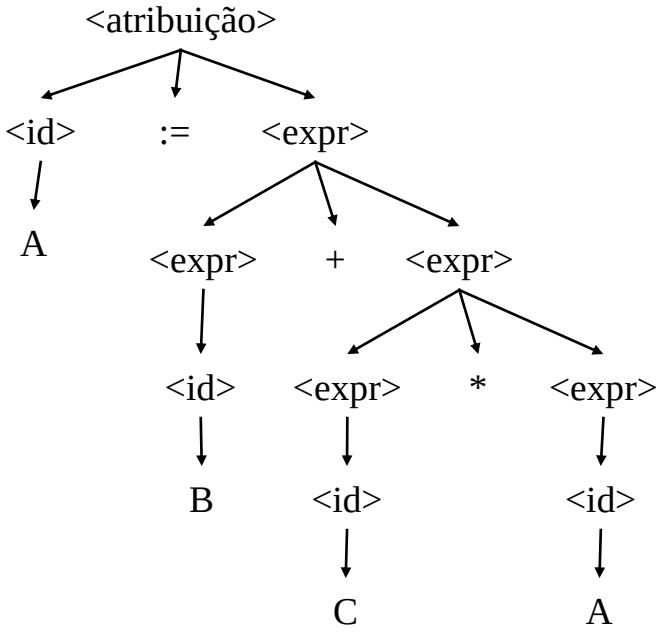
- **Sintaxe**
  - Árvores de análise (*parse trees*)
    - Descrevem a estrutura sintática hierárquica da linguagem

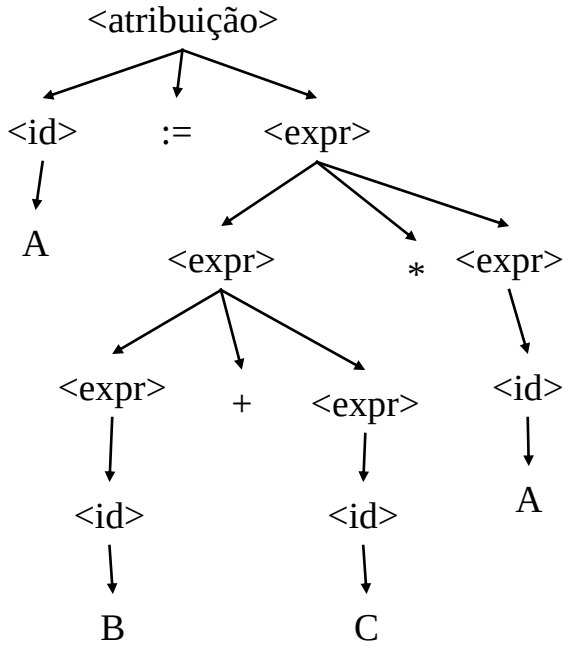
<atribuição> → <id> := <expr>
<id> → A | B | C
<expr> → <id> + <expr>
| <id> * <expr>
| (<expr>)
| <id>



- **Sintaxe**
  - Ambiguidade
    - Uma gramática é ambígua quando gera uma sentença para a qual há duas ou mais árvores de análise distintas

A := B + C * A


<atribuição>
<id>
:=
<expr>
A
<expr>
+
<expr>
<id>
<expr>
*
<expr>
B
<id>
<id>
C
A


<atribuição>
<id>
:=
<expr>
A
<expr>
*
<expr>
<expr>
+
<expr>
<id>
<id>
<id>
A
B
C

- **Sintaxe**
  - Exercício 2: mostre que a seguinte gramática é ambígua

```
<S> ::= <A>
<A> ::= <A> + <A> | <id>
<id> := a | b | c
```

- **Sintaxe**
  - Precedência de operadores
    - Uma gramática pode ser escrita para separar os operadores de adição e de multiplicação, a fim de que eles fiquem coerentemente dispostos na árvore de análise


```
<atribuição> ::= <id> := <expr>
<id> ::= A | B | C
<expr> ::= <expr> + <termo> | <termo>
<termo> ::= <termo> * <fator> | <fator>
<fator> ::= ( <expr> ) | <id>
```

- **Sintaxe**
  - BNF estendida
    - Objetivo: aumentar a legibilidade e a capacidade de escrita
  - Três extensões:
    - Utilização de colchetes
    - Utilização de chaves
    - Utilização de parênteses

- **Sintaxe**
  - BNF estendida
    - Utilização de colchetes

```
<selecao> → if (<expr>) <instrucao> [else <instrucao>]
```

- **Sintaxe**
  - BNF estendida
    - Utilização de colchetes

<selecao> → if (<expr>) <instrucao> [else <instrucao>]

    - Utilização de chaves

<lista_ident> → <identificador> { , <identificador}

- **Sintaxe**
  - BNF estendida
    - Utilização de colchetes

```
<selecao> → if (<expr>) <instrucao> [else <instrucao>]
```

    - Utilização de chaves

```
<lista_ident> → <identificador> { , <identificador}
```

    - Utilização de parênteses

```
<stmt_for> → for <var> := <expr> (to | downto) <expr> do
<stmt>
```

- **Sintaxe**
  - Exemplo BNF estendida

```
<expr> ::= <termo> + <expr>
         | <termo> - <expr> >
         | <termo>
<termo> ::= <fator> * <termo>
          | <fator> / <termo>
          |  <fator>
```

```
<expr> ::= <termo> { ( + | - ) <exp> }
<termo> ::= <fator> { ( * | / ) <termo> }
```

- **Sintaxe**
  - Desafio: incorpore à gramática abaixo, a abstração `<stmt_if>` e a defina de forma a refletir exatamente o funcionamento do comando IF na linguagem C.

```
<programa> → begin <lista_inst> end
<lista_inst> → <inst>
             | <inst>; <lista_inst>
<inst> → <var> = <expressao>
<var> → A | B | C
<expressao> → <var> + <var>
            |  <var> - <var>
            |  <var>
```